# Predicting Sample Collision with Neural Networks

Tuan Tran[1], Jory Denny[2], and Chinwe Ekenna[1]

*Abstract*— **Many state-of-art robotics applications require fast and efficient motion planning algorithms. Existing motion planning methods become less effective as the dimensionality of the robot and its workspace increases, especially the computational cost of collision detection routines. In this work, we present a framework to address the cost of expensive primitive operations in sampling-based motion planning. This framework determines the validity of a sample robot configuration through a novel combination of a Contractive AutoEncoder (CAE), which captures a point cloud representation of the robot's workspace, and a Multilayer Perceptron, which efficiently predicts the collision state of the robot from the CAE and the robot's configuration. We evaluate our framework on multiple planning problems with a variety of robots in 2D and 3D workspaces. The results show that (1) the framework is computationally efficient in all investigated problems, and (2) the framework generalizes well to new workspaces.**

## I. INTRODUCTION

Recently, machine learning has been infused in many solutions to notoriously difficult problems in robotics. These strategies have seen a wide array of success in visual sensing [1], task learning [2], and human-robot cooperation [3], etc. It is important to understand include learning in all levels of a robot's computation stack including common sub-components to solving these problems, e.g., subroutines for solving the motion planning problem.

In this work, we apply machine learning to predict the validity of robot configurations, a common procedure in sampling-based motion planners. Sampling-based approaches have seen broad applicability to many high-dimensional and complex motion planning problems in robotics [4], computer graphics [5], and computational biology [6]. Most of these approaches rely on the classification of robotic configurations into valid, e.g., collision-free, and invalid samples, while they work to construct graph approximations of a state space, called a roadmap. Thus, these approaches avoid full representation of the topology of the state space. However, as the dimensionality of the robot and the complexity of the workspace increase, the primitive operations of sampling-based methods become computationally obstructive [7].

Prior research has focused on improved geometric analysis techniques for collision checking [8], approximately modeling state space obstacles to predict the collision status of robot configurations, and lazily invoking the configuration

[1]Tuan Tran and Chinwe Ekenna are with the Department of Computer Science, University at Albany, SUNY, NY 12206, USA {ttran3, cekenna}@albany.edu.
[2]Jory Denny is with the Department of Mathematics and Computer Science, University of Richmond, VA 23173, USA jdenny@richmond.edu.

validation subroutine [4], among others. While these methods have shown success in many applications, none have fully and permanently bounded the computational cost of validating configurations in sampling-based motion planners.

We propose a novel framework to efficiently and effectively predict the validity of robot configurations in complex motion planning problems. Figure 1 shows the two parts of our framework: (1) Offline training phase: a Contractive AutoEncoder (CAE) [9], is trained to embed the robot's workspace into a latent space, and a Multilayer Perceptron (MLP), is trained to predict the validity of a robots' configurations given the CAE encoding of the workspace; (2) Online sampling validation phase: a workspace is encoded into latent space by CAE and its samples are validated by MLP using that encoded workspace. We examine the potential impact of such an approach with a variety of robots in complex 2D and 3D workspaces. For this work, we mainly focus on sampling because finding valid samples becomes paramount in narrow and cluttered passages. This work presents a first step towards applying our unique autoencoder variant to Sampling-based Motion Planning (SBMP). Our results indicate that: our framework is computationally efficient in all investigated problems, and our approach generalizes well to previously un-encoded workspaces. As such, our framework can be applied to many existing sampling-based routines to improve their computational efficiency.

## II. RELATED WORK

### A. Motion Planning Primitives

Let $X \subset \mathbb{R}^d$ denote a given state space, where $d \in \mathbb{N}$ is the dimensionality of the state space. The obstacle and obstacle-free state spaces are defined as $X_{obs} \subset X$ and $X_{free} = X \backslash X_{obs}$ respectively. A path $\tau$ for the robot is defined as a series of states $\{x_0, ..., x_n\}$, where n is the number of time steps in the path. A path is said to be feasible if it lies entirely in the obstacle-free space $X_{free}$

The motion planning problem is defined as: given a state space $X = X_{free} \subset X_{obs}$, an initial state $x_{init} \in X_{free}$, and a goal region $X_{goal} \subseteq X_{free}$, find a feasible path $\tau \in X_{free}$ such that $\tau_{start} = x_{init}$ and $\tau_{end} \in X_{goal}$. If no such path exists, report failure.

### B. Sampling-based Motion Planning (SBMP)

A common solution to the motion planning problem is the utilization of sampling-based methods. Most often, sampling-based motion planners randomly select and progressively expand and connect robot configurations to form approximate graph representations of $X_{free}$, called roadmaps. A roadmap encodes valid states as its nodes and transitions between them as its edges. In order to find a path, a starting state and goal

| Encoder: $f(X, W^e)$ | | Sample: S | Latent: Z | Label: $\theta$ | | Workspace: X | Sample: S |

Fig. 1: Overview of Proposed Framework

(a) Offline: Contractive AutoEncoder

$$\text{Encoder: } f(X, W^e)$$
$$\text{Latent: } Z = f(X, W^e)$$
$$\text{Decoder: } X' = g(Z, W^d)$$
$$L_{CAE} = \|X - X'\|^2 + \lambda \sum (W^e)^2$$

(b) Offline: Multilayer Perceptron

$$\text{Sample: S} \quad \text{Latent: Z} \quad \text{Label: } \theta$$
$$MLP(S, Z)$$
$$\text{Label: } \theta'$$
$$L_{MLP} = CrossEntropy(\theta, \theta')$$

(c) Online: Sample Validation

$$\text{Workspace: X} \quad \text{Sample: S}$$
$$CAE$$
$$MLP$$
$$\text{Label: } \theta$$
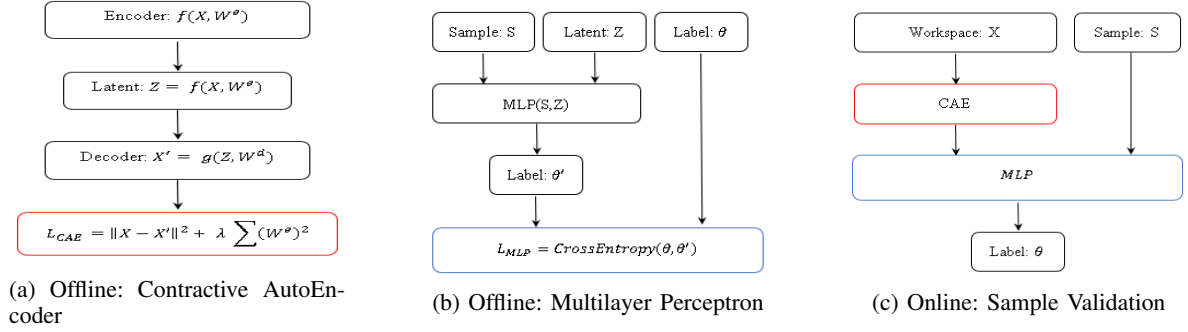
region are connected to the roadmap, and a feasible path is extracted from it.

These techniques employ the invocation of a "black box" collision detection module that classifies any state in either $X_{free}$ or $X_{obs}$ to avoid explicit construction of $X_{obs}$. This is used in the verification of both the nodes and the edges of any roadmap constructed using these methodologies. Commonly sampling-based strategies offer guarantees on probabilistic completeness and asymptotic optimality [10].

The main paradigms of sampling-based motion planning are the Probabilistic RoadMap [11] and the Rapidly-exploring Random Tree (RRT) algorithms [12]. Both PRM and RRT are known to be probabilistically complete, i.e., if a (robust) solution exists, then a solution will almost surely be found as the number of samples increases [7]. Many proposed variations on PRM and RRT improve their performance for a variety of scenarios. Inoue et al. [13] proposed a robot path planning method that combines RRT and long short-term memory network, which recalls the path of a robot by training with a large number of paths generated by RRT. Zhang et.al. [14] improved RRT algorithm by using a target bias sampling strategy, considering both distance and rotation angle when choosing the nearest neighbor, and making the planned path as smooth as possible using radial basis function neural network. Hauser proposed Lazy-PRM* [15] a lazy collision checking strategy, which avoids checking connections that have no chance to improve upon the current best path, thus reducing the per-sample computational cost and accelerating solution convergence. This work likewise aims to improve the performance of sampling-based approaches — specifically through reducing the cost of collision detection routines.

### C. Neural Network in Motion Planning

Representation learning [16] mainly aims to extract features from unstructured data, to either achieve a lower dimensional representation (often referred to as encoding) or learn features for supervised learning or reinforcement learning. There have been some recent works to utilize a learned low-dimensional latent model for motion planning. Ichter et al. [17] and Zhang et al [18] used the Conditional Variational Autoencoders to learn a non-uniform sampling methodology of a sampling-based motion planning algorithm. The work in [19] provides a new SBMP method using the learned latent space and plan motions directly in that space. Moreover,

Chen et al. [20] proposed a time-dependent variational autoencoder to learn dynamic motion primitives for humanoids, which could be applied to sampling configurations in motion planning. Qureshi et al. [21] learned a low-dimensional representation of the obstacle space by using a Contractive Autoencoder to encode a point cloud of the obstacles into a latent space and then used a feed-forward neural network to predict the next step an optimal planner would take given a start and goal. In this work, we aim to use a lower dimensional representation to reduce the cost of collision detection for use in motion planning algorithms.

The neural network is becoming prominent in path planning for its outstanding non-linear mapping ability. The shortcomings of neural network models include hardware requirements to reduce training time and difficulty of obtaining the best parameters for learning. However, with the fast development of hardware computing capabilities and undemanding requirements of path planning, those shortcomings are fading steadily. Li et al. [22] addressed a motion planning problem for vehicles with non-linear dynamics in a clustered environment using near-optimal RRT, which utilizes a neural network to estimate the cost function considering non-linear kinodynamic constraints. Moreover, Shi et al. [23] presented a motion planning system based on hybrid deep learning, which uses a convolutional neural network to reduce the dimension of the input image, a recurrent neural network to create a path tracking model, and a fully connected neural network to construct a control model. Chen et al. [24] proposed an optimal robot path planning system that builds a map, plans optimal paths, and maneuvers mobile robots, in which a simplified neural network is used to calculate the optimal trajectory for the robot. In this work, we make use of the multilayer perceptron to quickly and efficiently determine the validity of generated samples for the motion planning problem.

## III. RESEARCH FRAMEWORK

This section introduces our proposed framework. It is a neural network based motion planner comprised of two phases. The first phase trains a Contractive AutoEncoder and a Multilayer Perceptrons in an offline, apriori fashion. The second phase performs path finding computations online using the trained CAE and MLP.

## A. Offline Training

The proposed framework uses two neural network models to solve the motion planning problem. The first is a CAE that embeds a representation of the workspace, corresponding to a point cloud, into a latent space. The second is a MLP which predicts the validity of a given input sample and the CAE encoding of the workspace. Thus, instead of using the information of the whole workspace to decide whether a sample is valid or invalid, only the limited information from the encoded latent space is used.

*1) Contractive AutoEncoder (CAE):* The standard construction of a CAE consists of training neural networks for the encoder and the decoder. The encoder and the decoder are trained together using the reconstruction error. The output of the encoder represents a reduced representation of the initial input, and the decoder reconstructs that initial input from an encoded representation by minimizing a cost or loss function.

A CAE is used to embed workspace occupancy grids $X$ into an invariant and robust latent space $Z \subseteq \mathbb{R}^m$, where $m \in \mathbb{N}$ is the dimensionality of the latent space. Let $f(x, W^e)$ be an encoding function with weight matrix $W^e$ that encodes an input vector $x \in X$ into a vector in the latent space $z \in Z$. A decoding function $g(z, W^d)$, with weight matrix $W^d$, decodes a vector from the latent space $z \in Z$ back into a vector in the workspace $x \in X$. The objective function (loss function) for the CAE is:

$$L_{CAE} = \frac{1}{|D|} \sum_{x \in D} ||x - g(f(x, W^e), W^d)||^2 + \lambda \sum_{ij} (W_{ij}^e)^2 \tag{1}$$

where $\lambda$ is a penalizing coefficient and $D \subseteq X$ is the occupancy grid data for different workspaces. The penalizing term forces the latent space $z = f(x, W^e)$ to be contractive in the neighborhood of the training data which results in an invariant and robust feature learning [9]. Since we use linear encoder, the loss function of CAE is similar to the loss function of a regularized autoencoder. Moreover, our training data mainly consists of an unlabeled representation of the environment, we apply autoencoders to learn the low-dimensional representation of the training data instead of end-to-end architecture. By using the reconstruction (error) from the decoder, we could analyze a quantifiable metric used to decide which low-dimensional representation accurately replicates our environment the best.

*2) Multilayer Perceptrons (MLP):* We use a multilayer perceptron to perform sample validity. Given a workspace encoding $z = f(x, W^e) \in Z$, the samples' information, MLP predicts whether a sample is valid or not. The training objective for the MLP is to minimize the classification loss between the predicted sample's label and its actual label. Thus we use Cross-Entropy as our loss function since it's been shown to perform excellently for classification problems [25], [26].

## B. Online Sample Validation and Path Planning

The online phase exploits the neural models from the offline phase to do motion planning in cluttered and complex workspaces. Algorithm 1 presents the overall path generation procedure. First, using the offline-trained CAE, $W$ is encoded into a latent space $Z$. Then, a set of samples $S$ is generated based on the specification of a robot $R$ and its particular workspace $W$. Next, the offline-trained MLP takes the sample set $S$ and the encoded latent space $Z$ as input to determine a set of probably valid samples $S' \subseteq S$. Then, the probably valid sample set $S'$ is connected to create a roadmap $R$ by a proximity search function. If $R$ is not found, additional valid samples would be generated using auxiliary samplers. Finally, from $R$, a local planning function would return a feasible path between a given start and goal. We will discuss in more detail for each step in our algorithm in the following sections.

---

**Algorithm 1** SBMP Variant

---

**Input.** New workspace $X$ (point cloud data), Query $x_{init}$, $X_{goal}$

1: Use the CAE to encode $X$ into the latent space $Z$.
2: Randomly sample a set of states, $S$.
3: Feed $S$ and $Z$ into MLP, which predicts the validity of each sample in $S$. After this step $S'$ a set of most probably valid samples are retained.
4: Feed $S'$ into a standard PRM approach to yield a valid roadmap $R$. If $R$ is not found, create additional valid samples using auxiliary samplers.
5: Extract a path $\tau$ from $R$ between $x_{init}$ and $X_{goal}$.

---

*1) Sample Generation:* We use the following auxiliary samplers to generate the initial sample set $S$ from each workspace. Basic PRM (BS) [11] creates samples uniformly and randomly. Obstacle Based PRM (OB) [27] creates samples near the boundary of obstacles. Gaussian Sampler (G) [28] also creates samples around obstacles using adaptive probability and collision data. Bridge Test (BT) [29] creates samples using a short segment with two configurations and their midpoint.

*2) Workspace Encoder:* The encoding function $f(x, W^e)$, trained in the offline phase, is used to map the workspace occupancy grid $x \in X$ into a latent space $Z \subseteq \mathbb{R}^m$.

*3) Sample's Validity:* The MLP, a feed-forward neural network from the offline phase, takes the workspace encoding $Z$, the samples' information $S$, and predict samples' validity to create $S'$. In short, MLP is a classifier distinguishing between valid and invalid samples. To introduce stochasticity into the MLP and to prevent over-fitting, some of the hidden units in each hidden layer of the MLP were dropped out with a probability $p$. Dropout is applied layer-wise to a neural network and it drops each unit in the hidden layer with a probability $p : [0, 1] \in \mathbb{R}$.

*4) Connection and Path Planning:* We use a straight line local planner. Particularly a PRM approach is applied to $S'$ to generate a roadmap $R$, and the path is extracted and verified.

## IV. IMPLEMENTATION DETAILS

This section gives the implementation details of our framework. The proposed neural models, CAE and MLP, are

implemented in PyTorch [30].

### A. Data Collection

*1) Workspace Data:* To generate different workspace for training and testing, a number of quadrilateral blocks were placed in the operating region of $31 \times 31$ for 2D simple workspaces (2DS), $41 \times 41 \times 6$ for 3D office-like workspaces (3DO), and $11 \times 11 \times 11$ for 3D clutter workspaces (3DC). The placement of these blocks was randomly chosen in the operating region. Each random placement of the obstacle blocks leads to a different workspace. For each environment, the locations and numbers and types of obstacles are generated randomly. First, the number of obstacles is chosen randomly in a pre-defined range. Then for each obstacle in that environment, the type is chosen among the pre-defined types (small box, big box,..) and its location is also randomly determined. Currently, we do not consider any rotation of the blocks, i.e., they are axis aligned. Figure 2, 3, and 4 show some examples of our workspaces.

We represent the workspaces as an occupancy grid with values of $-1$ and $1$, in which $1$ means there is an obstacle at that point, and $-1$ means the point is in free space. The input to the encoder is an occupancy grid of size $n \times m$ for 2D and $n \times m \times k$ for 3D, where $n$, $m$, and $k$ are the number of points along each dimension of the workspace.

We use multiple workspace representations to train and test our CAE. The training data set of the 2D workspaces comprised of 30 workspaces, and for testing, two types of test data sets were created to evaluate the proposed method. The first test data set comprised of the 30 workspaces used in training, and the second test data set comprised of 10 previously unseen workspaces, i.e., 10 workspaces not from the training set. There are around 3 to 5 obstacle blocks with different shapes and sizes in each workspace.

The training data set of the 3D office-like workspaces comprised of 100 workspaces, and for testing 100 known workspaces and 20 previously unseen workspaces with 25 to 30 obstacle blocks in each workspace were used.

The training data set of the 3D clutter workspaces comprised of 50 workspaces, and for testing 30 known workspaces and 10 previously unseen workspaces with 110 to 125 same shape and size obstacle blocks in each workspace were used.

To test the performance of our MLP, we generate multiple different workspaces with different starts and goals. For 2D simple and 3D office-like, we randomly generate 10 different workspaces with 10 different start and goal positions for each of them; and 100 3D clutter workspaces with 1 different start and goal positions for each of them to test the performance of our proposed framework in terms of time.

*2) Sample Data:* For the 2D simple workspace, we use a point-mass robot with 2 degrees of freedom (DoF) that represents its $\{x, y\}$ position. The point robot is (0.5x0.5) in a (31x31) environment. For the 3D office-like, we use KuKa robot [31], which has 5 joints, with a total of 8 DoF. The 8 DoFs represent the robot's $\{x, y\}$ position, rotation, and each joint's rotation. The Kuka robot is the size of (1.5x1x1) in a (41x41x6) environment. For the 3D clutter workspaces, we
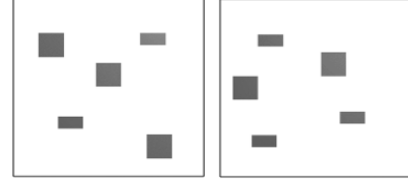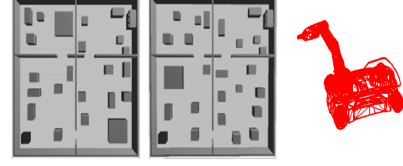


Fig. 2: Examples of 2D simple



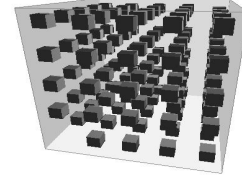Fig. 3: Examples of 3D office-like workspace with Kuka robot



Fig. 4: Example of 3D clutter workspace

use box robots with 7 DoF and 9 DoF. The DoFs represent the robot's $\{x, y, z\}$ position, $\{\alpha, \beta, \gamma\}$ rotation, and each joint's rotation. The box robots consist of 2 types of links: a small link of (0.1x0.1x0.1) and a big link of (0.4x0.1x0.1) in a (11x11x11) environment. The 7D box robot has 1 joint for 2 links, and the 9D box robot has 3 joints for 2 small links and 2 big links. We generate 100 samples for each 2D workspace and 200 samples for each 3D workspace for training and testing our MLP separately. This is evaluated as seen scenarios. Those samples are approximately distributed 50/50 between valid and invalid samples.

Since our approach mainly focuses on verifying the sample's validity, we incorporate it into the standard PRM approaches using algorithm 1 to test its effectiveness. However, instead of having a large set of samples $S'$, we provide a small set of samples $S'$ and let the algorithm decide whether those supplied samples are enough or there is a need to generate more samples. We think that providing the algorithm with too many samples would negatively impact the performance of neighborhood finding routines in sampling-based planners. Thus, we generate 200 samples regardless of their validity for each 2D simple workspace, 2000 samples for each 3D office-like workspace, and 400 samples for each 3D clutter workspace. This is evaluated as unseen scenarios

### B. Model Architecture

*1) Contractive AutoEncoder:* Since the structure of the decoders is the inverse of the encoder, we only describe the structure of the encoder.

For the 2D workspaces, the encoding function $f(x, W^e)$ and decoding function $g(f(x, W^e), W^d)$ consist of five linear layers and one output layer, where each layer uses

Parametric Rectified Linear Unit (PReLU) [32] as its activation function, for loss function, we use mean square error with weight-decay. The layers 1 to 5 transforms the input vector $31 \times 31$ to 512, 256, 128, 64, and 32 hidden units, respectively. The output layer takes 32 units as input and outputs 12 units. Hence the workspace representation in the latent space is a vector of size 12. Then, those 12 units are used by the decoder to reconstruct the $31 \times 31$ space.

For the 3D office-like workspaces, the encoding function $f(x, W^e)$ and decoding function $g(f(x, W^e), W^d)$ are the same as in 2D simple workspace but consist of seven linear layers and one output layer. Layers one to seven transform the input vectors to 5043, 3125, 1600, 800, 400, 200, 100 hidden units, respectively. The output layer takes the 100 units from the seventh layer and transforms them into 50 units. For the 3D clutter workspaces, layers one to six transform the input vectors to 1000, 800, 600, 400, 200, 100 hidden units, the output layer transforms the input 100 units to output 50 units.

*2) Multi-layer Perceptrons:* The input is given by concatenating the encoded workspace's representation $Z$, and the DoF for each robot from a given state. Each of the layers is a sandwich of a linear layer, a Parametric Rectified Linear Unit (PReLU), and Dropout ($p$).

For the 2D workspaces, MLP is a 4-layer neural network. Layers one and two map the input vectors to 6, 4 units, respectively. The output layer takes the 4 units and transforms them into the validity of each sample, which is either valid or invalid. For the 3D workspaces, MLP is a 7-layer neural network. Layers one to six map the input vectors to 50, 40, 30, 20, 10, 5 units, respectively. The output layer takes the 5 units from the sixth layer and transforms them into the validity of each sample.

*3) Parameters:* To train the neural models CAE and MLP for both 2D and 3D workspace, the Adagrad [33] optimizer was used with the learning rate of $0.1$. The Dropout probability $p$ and penalizing term $\lambda$ were set to $0.5$ and $0.001$, respectively.

## V. RESULTS

This section presents the results of the framework for the motion planning of a point robot in the 2D workspaces and the KuKa robot and a box robot in 3D workspaces.

Table I shows the performance for the CAE. For 2DS, the average accuracy of our CAE is $100\%$ with the variance of $0.4\%$ for both already seen workspaces and unseen workspaces. For 3DO, the average accuracy of our CAE is $96\%$ with the variance of $3\%$ for already seen workspaces and the average accuracy is $94\%$ with the variance of $5\%$ for unseen workspaces. The accuracy and variance are $98\%$ and $1\%$ and $95\%$ and $3\%$ for seen and unseen 3D clutter, respectively. Overall, the accuracy of our CAE is excellent. Thus, we are able to learn the underlying structure of the workspaces quite well.

|  | Seen | Unseen |
|---|---|---|
| 2DS | 100% | 100% |
| 3DO | 96% | 94% |
| 3DC | 98% | 95% |

TABLE I: Performance of CAE

|  | Seen | | | Unseen | | |
|---|---|---|---|---|---|---|
|  | Acc | TPR | TNR | Acc | TPR | TNR |
| 2DS | 92% | 96% | 88% | 88% | 90% | 85% |
| 3DO | 76% | 88% | 64% | 73% | 85% | 60% |
| 3DC-7 | 75% | 87% | 64% | 71% | 85% | 61% |
| 3DC-9 | 73% | 82% | 66% | 72% | 82% | 62% |

TABLE II: Accuracy, True Positive Rate, and True Negative Rate of MLP

Table II shows the accuracy, true positive rate (TPR), and true negative rate (TNR) for the MLP, with false negative rate = 1–TPR and false positive rate = 1–TNR. For 2D workspaces, the average accuracy of our MLP is $92\%$ with the variance of $2\%$ for already seen workspaces and the average accuracy is $88\%$ with the variance of $4\%$ for unseen workspaces. For 3DO workspace, the average accuracy of our MLP is $76\%$ with the variance of $6\%$ for already seen workspaces, and the average accuracy is $73\%$ with the variance of $8\%$ for unseen workspaces. The average accuracy is $73\%$ with $5\%$ variance (seen), and $71\%$ with $9\%$ variance (unseen) for 3DC-7. The average accuracy is $73\%$ with $6\%$ variance (seen), and $72\%$ with $10\%$ variance (unseen) for 3DC-9. Additionally, the true positive rates and true negative rates are quite good for our MLP, thus most of the valid samples and invalid samples are successfully validated by MLP. Overall, the accuracy of our MLP is acceptable.

|  | Sampling Time | | | | Total Time | | | |
|---|---|---|---|---|---|---|---|---|
|  | BS | OB | G | BT | BS | OB | G | BT |
| 2DS | 53% | 57% | 63% | 74% | 17% | 18% | 11% | 8% |
| 3DO | 45% | 58% | 41% | 56% | 1% | 9% | 10% | -8% |
| 3DC-7 | 49% | 58% | 51% | 61% | -2% | 14% | -9% | 54% |
| 3DC-9 | 39% | 59% | 48% | 62% | 18% | 20% | 13% | 37% |

TABLE III: Average time improvement ratio with and without applying our proposed framework for sampling time and total time.

Table III show the average time improvement ratio comparison for sampling time and total time with and without applying our method with Basic PRM (BS), Obstacle Based PRM (OB), Gaussian (G), and Bridge Test (BT) for solving motion planning problems.

In 2D workspaces, the average time to validate samples without and with using our method are: $0.0004 \pm 0.0002$ and $0.0002 \pm 0.0002$ for BS, $0.05 \pm 0.01$ and $0.03 \pm 0.01$ for OB, $0.04 \pm 0.01$ and $0.02 \pm 0.01$ for G, $0.26 \pm 0.23$ and $0.11 \pm 0.03$ for BT. There are significant time improvement when using our method for classify valid samples. Moreover, the average total execution time for each query without and with using our method are: $0.12 \pm 0.01$ and $0.10 \pm 0.01$ for BS, $0.38 \pm 0.05$ and $0.31 \pm 0.03$ for OB, $0.09 \pm 0.01$ and $0.09 \pm 0.01$ for G, $0.13 \pm 0.01$ and $0.13 \pm 0.01$ for BT. Notable, our framework has good performance. Additionally, we notice there is a nearly $20\%$ improvement when applying our framework to OB.

In 3DO workspaces, the average time to validate samples without and with using our method are: $0.11 \pm 0.01$ and $0.06 \pm 0.02$ for BS, $1.13 \pm 0.08$ and $0.56 \pm 0.25$ for OB, $0.21 \pm 0.04$ and $0.12 \pm 0.02$ for G, $0.76 \pm 0.13$ and $0.32 \pm 0.02$ for BT. The time improvements when using our method for
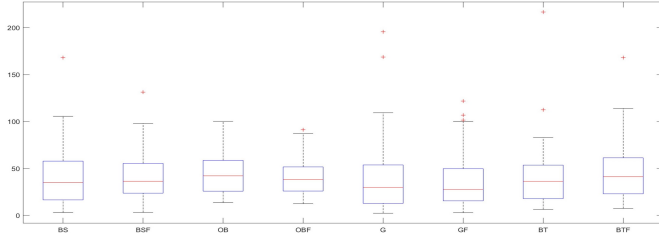
Fig. 5: Time performance with and without applying our method in 3DO workspaces
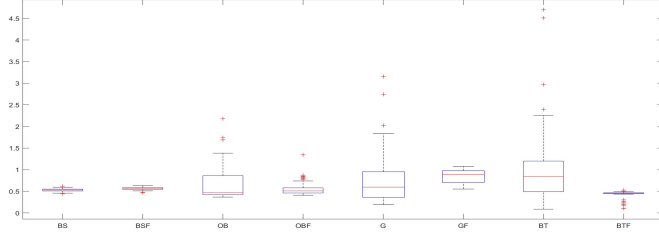


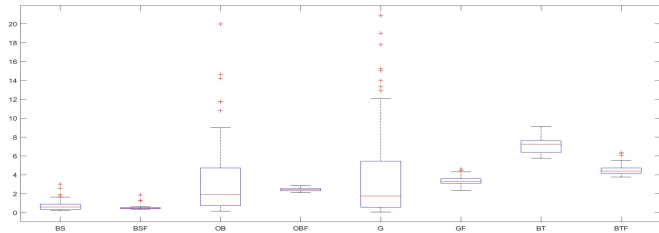Fig. 6: Time performance with and without applying our method in 3DC-7 workspaces



Fig. 7: Time performance with and without applying our method in 3DC-9 workspaces

classifying valid samples are quite good. Additionally, the average total execution time for each query without and with using our method are: $40.86 \pm 36.47$ and $40.51 \pm 25.91$ for BS, $43.74 \pm 22.01$ and $39.88 \pm 18.72$ for OB, $42.32 \pm 41.99$ and $38.22 \pm 30.01$ for G, $40.40 \pm 33.92$ and $43.13 \pm 28.87$ for BT. There are improvements when using our framework with OB (9%) and G (10%), but there is a small improvement for BS (1%) and a slow-down for BT (8%). We suspect that it could be that the supplied samples are not good enough, thus there is the need for re-sampling.

The average time to validate samples without and with using our method for 3DC-7 are: $0.007 \pm 0.003$ and $0.003 \pm 0.001$ for BS, $0.20 \pm 0.03$ and $0.09 \pm 0.01$ for OB, $0.09 \pm 0.01$ and $0.04 \pm 0.02$ for G, $0.42 \pm 0.04$ and $0.17 \pm 0.02$ for BT; and for 3DC-9 are: $0.015 \pm 0.005$ and $0.008 \pm 0.003$ for BS, $0.08 \pm 0.01$ and $0.04 \pm 0.01$ for OB, $0.04 \pm 0.01$ and $0.02 \pm 0.01$ for G, $0.10 \pm 0.02$ and $0.04 \pm 0.02$ for BT. Overall, our method reduces the sampling classification time notably. Moreover, the average total execution time for each query without and with using our method for 3DC-7 are: $0.53 \pm 0.03$ and $0.54 \pm 0.03$ for BS, $0.65 \pm 0.35$ and $0.56 \pm 0.13$ for OB, $0.45 \pm 0.40$ and $0.49 \pm 0.12$ for G, $0.95 \pm 0.64$ and $0.44 \pm 0.05$ for BT; and for 3DC-9 are: $0.73 \pm 0.48$ and $0.60 \pm 0.19$ for BS, $3.27 \pm 3.62$ and $2.60 \pm 0.19$ for OB, $3.90 \pm 4.73$ and $3.40 \pm 0.41$ for G, $7.10 \pm 0.77$ and $4.50 \pm 0.50$ for BT. There are quite notable time improvements when using

our framework to augment the motion planning algorithms for 3DC workspaces. In 3DC workspaces, the BT samplers perform the worst because of the open nature of our testing workspaces.

Figure 5, 6, 7 show the performance with (BSF, OBF, GF, BTF) and without (BS, OB, G, BT) applying our proposed framework in second for 3DO, 3DC-7, 3DC-9 workspace. Overall, there is a notable improvement in performance in most cases. The standard deviations when using our framework decreased compared to not using it. Thus, our framework augments the algorithms perfectly and makes them more stable in solving queries reliably. Although we only test our approach using fixed size workspaces, it is robust to varying sizes. By splitting the workspace into parts with a pre-defined size, the proposed approach would still be able to validate samples. Moreover, by adjusting the number of samples in the sampling validation phase and the number of samples generated by the auxiliary sampler, our method guarantees probabilistic completeness for any motion planning problem. Therefore, our approach is applicable to most of sampling based planners.

## VI. CONCLUSION

In this paper, we present a fast and efficient neural network framework for sampling-based motion planners. The framework consists of a Contractive AutoEncoder that encodes a point cloud representation of a robot's workspace into a latent feature space and a feed-forward neural network that takes that encoded workspace and robot configuration details to predict the validity of that configuration. The main advantage of our framework is that we use an encoded workspace and neural network to reduce or even remove expensive collision detection operations. In this paper, we focus on the sampling phase of SBMP and how we can reduce number of collision checks made. Our proposed method is quite robust such that it can be integrated into any sampling strategy.

In the future, we plan to research models that increase the accuracy of our MLP by pre-processing a configuration's information before training. Furthermore, since our method considers the environment's and robot's information when performing collision checks, we could train them on regions of the environments. Thus, instead of using our method for the whole environment, an SBMP algorithm could use our method when it reaches certain potions of that environment. Additionally, we could apply this approach to other aspects of SBMP, e.g., edge prediction, and path validation.

## REFERENCES

[1] B. Espiau, F. Chaumette, and P. Rives, "A new approach to visual servoing in robotics," *ieee Transactions on Robotics and Automation*, vol. 8, no. 3, pp. 313–326, 1992.

[2] C. L. Nehaniv and K. E. Dautenhahn, *Imitation and social learning in robots, humans and animals: behavioural, social and communicative dimensions.* Cambridge University Press, 2007.

[3] P. Trautman, J. Ma, R. M. Murray, and A. Krause, "Robot navigation in dense human crowds: Statistical models and experimental studies of human–robot cooperation," *The International Journal of Robotics Research*, vol. 34, no. 3, pp. 335–356, 2015.

[4] J. Denny, K. Shi, and N. M. Amato, "Lazy toggle prm: A single-query approach to motion planning," in *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 2407–2414.

[5] L. Kobbelt and M. Botsch, "A survey of point-based techniques in computer graphics," *Computers & Graphics*, vol. 28, no. 6, pp. 801–814, 2004.

[6] C. Ekenna, S. Thomas, and N. M. Amato, "Adaptive local learning in sampling based motion planning for protein folding," *BMC systems biology*, vol. 10, no. 2, p. 49, 2016.

[7] J. Bialkowski, S. Karaman, M. Otte, and E. Frazzoli, "Efficient collision checking in sampling-based motion planning," in *Algorithmic Foundations of Robotics X*. Springer, 2013, pp. 365–380.

[8] M. Lin and S. Gottschalk, "Collision detection between geometric models: A survey," in *Proc. of IMA conference on mathematics of surfaces*, vol. 1, 1998, pp. 602–608.

[9] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio, "Contractive auto-encoders: Explicit invariance during feature extraction," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*. Omnipress, 2011, pp. 833–840.

[10] B. Ichter and M. Pavone, "Robot motion planning in learned latent spaces," *arXiv preprint arXiv:1807.10366*, 2018.

[11] L. Kavraki, P. Svestka, and M. H. Overmars, *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*. Unknown Publisher, 1994, vol. 1994.

[12] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[13] M. Inoue, T. Yamashita, and T. Nishida, "Robot path planning by lstm network under changing environment," in *Advances in Computer Communication and Computational Sciences*. Springer, 2019, pp. 317–329.

[14] P. Zhang, C. Xiong, W. Li, X. Du, and C. Zhao, "Path planning for mobile robot based on modified rapidly exploring random tree method and neural network," *International Journal of Advanced Robotic Systems*, vol. 15, no. 3, p. 1729881418784221, 2018.

[15] K. Hauser, "Lazy collision checking in asymptotically-optimal motion planning," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 2951–2957.

[16] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[17] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 7087–7094.

[18] C. Zhang, J. Huh, and D. D. Lee, "Learning implicit sampling distributions for motion planning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 3654–3661.

[19] B. Ichter and M. Pavone, "Robot motion planning in learned latent spaces," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2407–2414, 2019.

[20] N. Chen, M. Karl, and P. van der Smagt, "Dynamic movement primitives in latent space of time-dependent variational autoencoders," in *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*. IEEE, 2016, pp. 629–636.

[21] A. H. Qureshi, M. J. Bency, and M. C. Yip, "Motion planning networks," *arXiv preprint arXiv:1806.05767*, 2018.

[22] Y. Li, R. Cui, Z. Li, and D. Xu, "Neural network approximation based near-optimal motion planning with kinodynamic constraints using rrt," *IEEE Transactions on Industrial Electronics*, vol. 65, no. 11, pp. 8718–8729, 2018.

[23] C. Shi, X. Lan, and Y. Wang, "Motion planning for unmanned vehicle based on hybrid deep learning," in *2017 International Conference on Security, Pattern Analysis, and Cybernetics (SPAC)*. IEEE, 2017, pp. 473–478.

[24] Y.-W. Chen and W.-Y. Chiu, "Optimal robot path planning system by using a neural network-based approach," in *2015 international automatic control conference (CACS)*. IEEE, 2015, pp. 85–90.

[25] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for youtube recommendations," in *Proceedings of the 10th ACM conference on recommender systems*. ACM, 2016, pp. 191–198.

[26] T. N. Sainath, B. Kingsbury, and B. Ramabhadran, "Auto-encoder bottleneck features using deep belief networks," in *2012 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2012, pp. 4153–4156.

[27] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo, "Obprm: An obstacle-based prm for 3d workspaces," 1998.

[28] V. Boor, M. H. Overmars, and A. F. Van Der Stappen, "The gaussian sampling strategy for probabilistic roadmap planners," in *ICRA*, 1999, pp. 1018–1023.

[29] D. Hsu, T. Jiang, J. Reif, and Z. Sun, "The bridge test for sampling narrow passages with probabilistic roadmap planners," in *2003 IEEE international conference on robotics and automation (cat. no. 03CH37422)*, vol. 3. IEEE, 2003, pp. 4420–4426.

[30] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[31] K. R. Corporation, "Kuka youbot," www.youbot-store.com, accessed: June 1, 2013.

[32] L. Trottier, P. Gigu, B. Chaib-draa *et al.*, "Parametric exponential linear unit for deep convolutional neural networks," in *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2017, pp. 207–214.

[33] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.